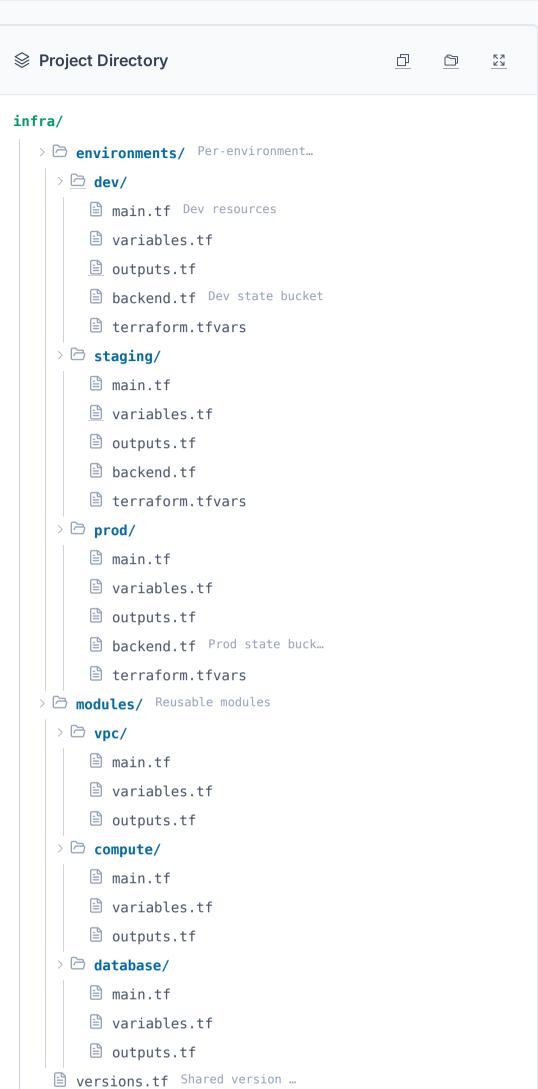
> Terraform Multi-Environment Project Structure

Separate dev/staging/prod environments. Directory-based isolation with shared modules.

#terraform #iac #devops #multi-env #workspaces



○ Why This Structure?

Directory-based environment separation is safer than workspaces for production. Each environment has its own state file and backend config. Shared modules in modules/ prevent drift between environments while allowing different variable values.

environments/ - One folder per environment, isolated state
environments/*/backend.tf - Separate state bucket per environment
modules/ - Shared modules referenced via relative path
terraform.tfvars - Environment-specific values

</> Module Usage

```
# environments/dev/main.tf
module "vpc" {
    source = "../../modules/vpc"
    cidr = var.vpc_cidr
    env = "dev"
}

module "compute" {
    source = "../../modules/compute"
    vpc_id = module.vpc.vpc_id
    instance_type = "t3.small" # Smaller in dev
}
```

Getting Started

- Create environments/dev/ structure
- 2. Build shared modules in modules/
- 3. cd environments/dev && terraform init
- 4. terraform plan
- 5. Repeat for staging and prod

☑ When To Use This

- Need isolated dev/staging/prod
- Different cloud accounts per environment
- Want to promote changes through environments
- Multiple team members working on infra
- Compliance requires environment isolation

of Trade-offs

Duplication - Some config repeated across environments

Module sync - Must update all envs when module changes

Directory navigation - More cd commands during development

☑ Best Practices

- Never share state files between environments
- Use different AWS accounts for prod vs non-prod
- Pin module versions for production stability
- Keep environment differences in .tfvars only
- Use CI/CD to promote changes dev → staging → prod